

**Name Mehran Ali Shah**  
**Id No 13943**  
**Paper Visual Programming**  
**Date-23-09-2020**

**Question No 1:**

**(a)**

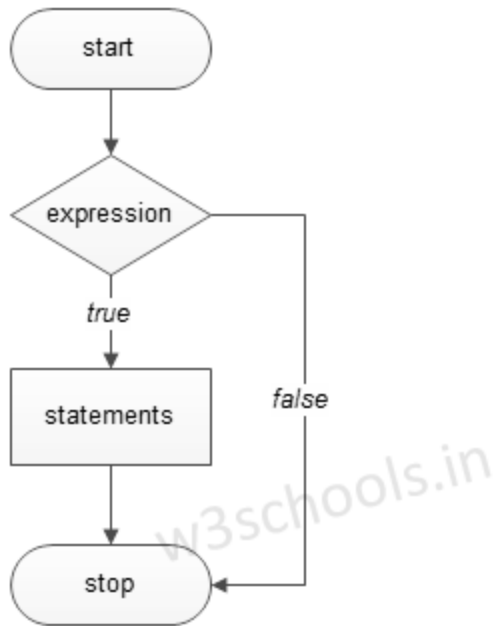
**Answer:**

**Decision Making:**

As we all make decisions in our real life, similarly in the logical world of programming, decisions are an essential part of executing a specific block of code based on the fulfillment of the condition. The control statements control the flow of statements, and based on certain conditions; different logical blocks are executed. The programming language provides all these things as a form of conditional statements or decision-making statements.

These types of statements are used by programmers to determine one or more conditions evaluated by the program at run-time. Specific blocks of code associated with these statements will be executed only when the condition is determined.

**Flow Chart:**



**(b)**

Answer:

```
using System;
```

```
class Program
```

```
{
```

```
    static void Main(string[] args)
```

```
    {
```

```
        char gender;
```

```
        //Reading gender from user
```

```
Console.WriteLine("Enter gender (M/m or F/f): ");
gender = Convert.ToChar(Console.ReadLine());

// checking vowel and consonant
switch (gender)
{
    case 'M':
        case 'm': Console.WriteLine("MALE");
        break;

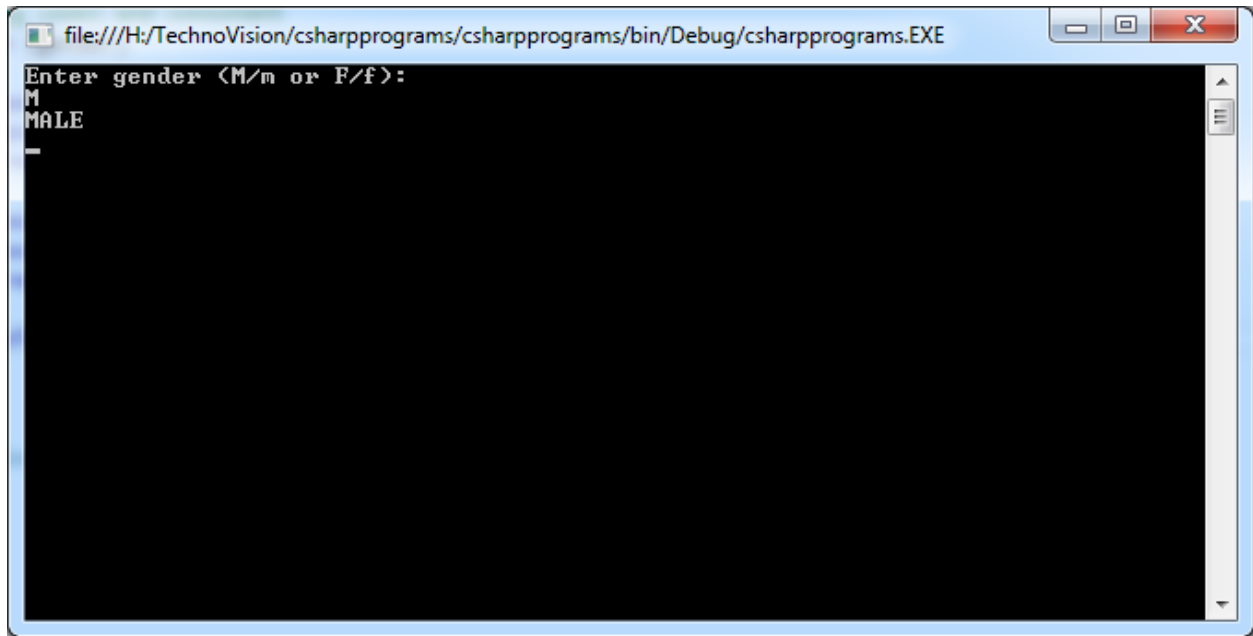
    case 'F':
        case 'f': Console.WriteLine("FEMALE");
        break;

    default: Console.WriteLine("Unspecified Gender");
        break;
}

Console.ReadLine();
}

}
```

**Output:**

A screenshot of a Windows console window titled "file:///H:/TechnoVision/csharpprograms/csharpprograms/bin/Debug/csharpprograms.EXE". The window has a black background with white text. The prompt "Enter gender (M/m or F/f):" is displayed at the top. Below it, the user has entered "M", and the program has output "MALE".

```
file:///H:/TechnoVision/csharpprograms/csharpprograms/bin/Debug/csharpprograms.EXE
Enter gender (M/m or F/f):
M
MALE
```

## Question No 2:

(a)

**Answer:**

**If-else-if:**

In c#, if-else-if statement or condition is used to define multiple conditions and execute only one matched condition based on our requirements. Generally, in c# if statement or if-else statement is useful when we have one condition to validate and execute the required block of statements. In case, if we have multiple conditions to validate and execute only one block of code, then the if-else-if statement is useful in our application.

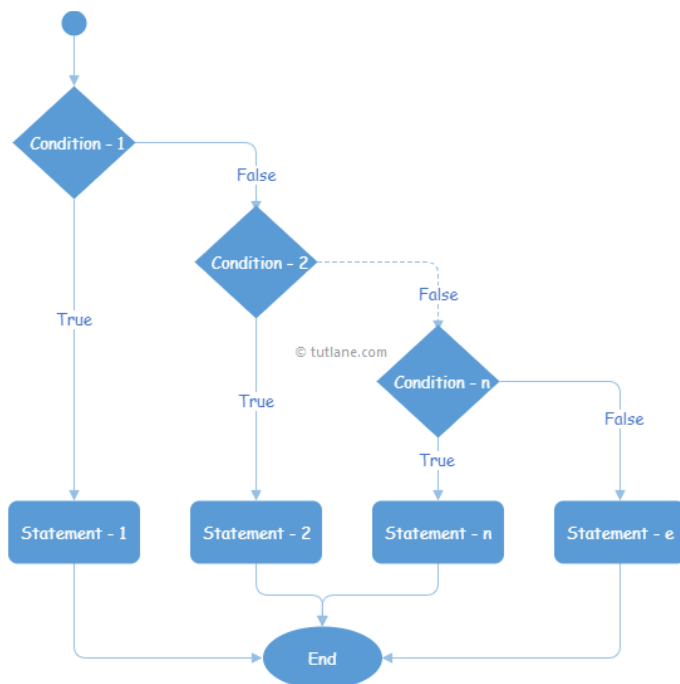
Syntax:

```
if(first-condition) {
```

```
// when first condition becomes true
```

```
}  
else if(second-condition)  
{  
// when second condition becomes true  
}  
else if(third-condition)  
{  
// when third condition becomes true  
}  
else  
{  
// when all the conditions are false  
}
```

**Flow Chart:**



**(b)**

**Answer:**

Temp < 0 then Freezing weather

Temp 0-10 then Very Cold weather

Temp 10-20 then Cold weather

Temp 20-30 then Normal in Temp

Temp 30-40 then Its Hot

Temp >=40 then Its Very Hot

**Program**

```
using System;
```

```
public class Exercise13
```

```
{
```

```
    public static void Main()
```

```
{
```

```
    int tmp;
```

```
    Console.Write("\n\n");
```

```
    Console.Write("Accept a temperature in centigrade and display a  
suitable message:\n");
```

```
Console.Write("-----  
---");
```

```
Console.Write("\n\n");
```

```
Console.Write("Input days temperature : ");
```

```
tmp= Convert.ToInt32(Console.ReadLine());
```

```
if(tmp<0)
```

```
    Console.Write("Freezing weather.\n");
```

```
else if(tmp<10)
```

```
    Console.Write("Very cold weather.\n");
```

```
else if(tmp<20)
```

```
    Console.Write("Cold weather.\n");
```

```
else if(tmp<30)
```

```
    Console.Write("Normal in temp.\n");
```

```
else if(tmp<40)
```

```
    Console.Write("Its Hot.\n");
```

```
else
```

```
    Console.Write("Its very hot.\n");
```

```
}
```

```
}
```

Sample Output:

Accept a temperature in centigrade and display a suitable message:

---

Input days temperature : 35

Its Hot.

### Question No 3:

(a)

**Answer:**

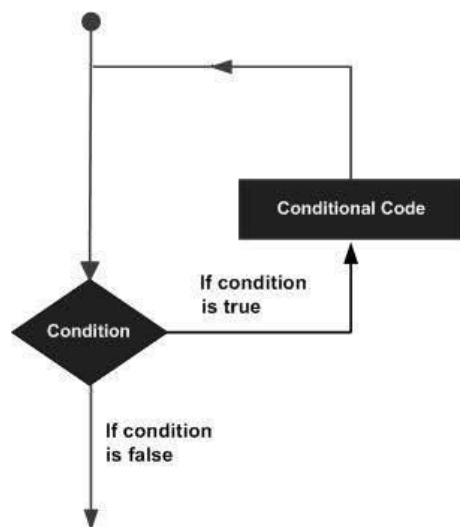
**Loops:**

Looping in a programming language is a way to execute a statement or a set of statements multiple times depending on the result of the condition to be evaluated to execute statements. The result condition should be true to execute statements within loops.

Loops are mainly divided into two categories:

Entry Controlled Loops: The loops in which condition to be tested is present in beginning of loop body are known as Entry Controlled Loops. while loop and for loop are entry controlled loops.

**Flow Chart:**





(b)

**Answer:**

**while loop:**

The test condition is given in the beginning of the loop and all statements are executed till the given boolean condition satisfies when the condition becomes false, the control will be out from the while loop.

**Syntax:**

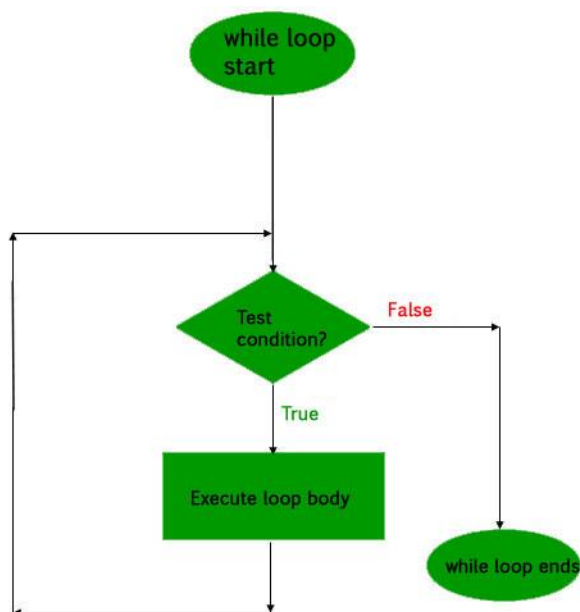
```
while (boolean condition)
```

```
{
```

```
    loop statements...
```

```
}
```

**Flow Chart:**



**Program:**

```
// C# program to illustrate while loop  
using System;
```

```
class whileLoopDemo  
{  
    public static void Main()  
    {  
        int x = 1;  
  
        // Exit when x becomes greater than 4  
        while (x <= 4)  
        {  
            Console.WriteLine("Visual Programming");  
  
            // Increment the value of x for  
            // next iteration  
            x++;  
        }  
    }  
}
```

**Output:**

Visual Programming

Visual Programming

Visual Programming

Visual Programming

### **for loop:**

for loop has similar functionality as while loop but with different syntax. for loops are preferred when the number of times loop statements are to be executed is known beforehand. The loop variable initialization, condition to be tested, and increment/decrement of the loop variable is done in one line in for loop thereby providing a shorter, easy to debug structure of looping.

**Initialization of loop variable:** The expression / variable controlling the loop is initialized here. It is the starting point of for loop. An already declared variable can be used or a variable can be declared, local to loop only.

**Testing Condition:** The testing condition to execute statements of loop. It is used for testing the exit condition for a loop. It must return a boolean value true or false. When the condition became false the control will be out from the loop and for loop ends.

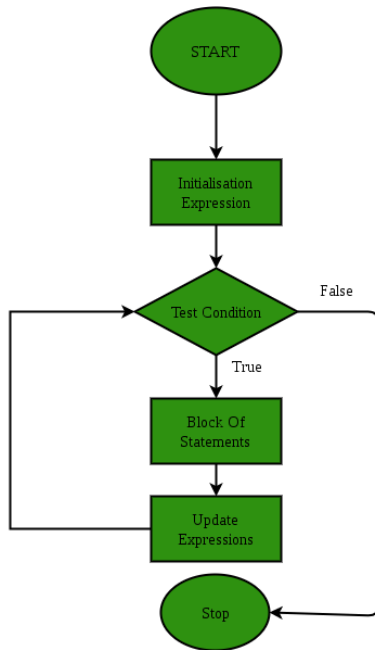
**Increment / Decrement:** The loop variable is incremented/decremented according to the requirement and the control then shifts to the testing condition again.

### **Syntax:**

```
for (loop variable initialization ; testing condition;  
      increment / decrement)  
{  
    // statements to be executed
```

```
}
```

## Flow Chart:



## Program:

```
// C# program to illustrate for loop.  
using System;
```

```
class forLoopDemo
```

```
{
```

```
    public static void Main()
```

```
    {
```

```
        // for loop begins when x=1
```

```
        // and runs till x <=4
```

```
        for (int x = 1; x <= 4; x++)
```

```
            Console.WriteLine("Visual Programming");
```

```
    }  
}
```

### **Output:**

Visual Programming

Visual Programming

Visual Programming

Visual Programming

### **do-while loop:**

do while loop is similar to while loop with the only difference that it checks the condition after executing the statements, i.e it will execute the loop body one time for sure because it checks the condition after executing the statements.

### **Syntax:**

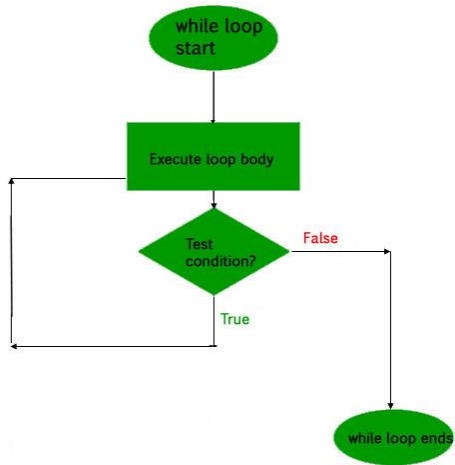
```
do
```

```
{
```

```
    statements..
```

```
}while (condition);
```

### **Flow Chart:**



### Program:

// C# program to illustrate do-while loop  
using System;

```
class dowhileloopDemo  
{  
    public static void Main()  
    {  
        int x = 21;  
        do  
        {  
            // The line will be printed even  
            // if the condition is false  
            Console.WriteLine("Visual Programming");  
            x++;  
        }  
    }  
}
```

```
    }  
    while (x < 20);  
    }  
}
```

## **Output:**

Visual Programming

## **Infinite Loops:**

The loops in which the test condition does not evaluate false ever tend to execute statements forever until an external force is used to end it and thus they are known as infinite loops.

## **Program:**

```
// C# program to demonstrate infinite loop  
using System;
```

```
class infiniteLoop
```

```
{  
    public static void Main()  
    {  
        // The statement will be printed  
        // infinite times  
        for(;;)  
            Console.WriteLine("This is printed infinite times");  
    }  
}
```

```
}
```

**Output:**

This is printed infinite times  
This is printed infinite times  
This is printed infinite times  
This is printed infinite times  
This is printed infinite times  
This is printed infinite times  
This is printed infinite times  
This is printed infinite times  
.....

**Question No 4:**

**Answer:**

Because programmers used counting, or indexing variables like this so often, the for loop was created to simplify the process. Essentially a for loop is used to iterate through a block of code a specified number of times, with the use of an iterative variable

**Program:**

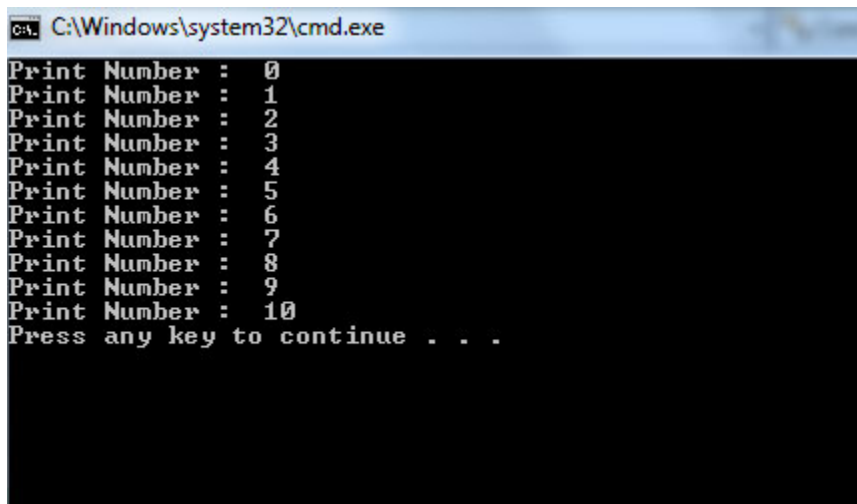
```
class Program
{
    static void Main(string[] args)
    {
        for(int k =0; k <= 10; k++)
        {
```



```
Console.WriteLine("Print Number : {0}",k);
```

```
    }  
    }  
}
```

## Output:



```
CA: C:\Windows\system32\cmd.exe  
Print Number : 0  
Print Number : 1  
Print Number : 2  
Print Number : 3  
Print Number : 4  
Print Number : 5  
Print Number : 6  
Print Number : 7  
Print Number : 8  
Print Number : 9  
Print Number : 10  
Press any key to continue . . .
```

## Question No 5:

(a)

### Answer:

Encapsulation is defined 'as the process of enclosing one or more items within a physical or logical package'. Encapsulation, in object oriented programming methodology, prevents access to implementation details.

Abstraction and encapsulation are related features in object oriented programming. Abstraction allows making relevant information visible and encapsulation enables a programmer to implement the desired level of abstraction. Encapsulation is implemented by using access specifiers. An access specifier defines the scope and visibility of a class member.

C# supports the following access specifiers –

Public

Private

Protected

Internal

Protected internal

**(b)**

**Answer:**

In C#, you can manage encapsulation with access modifiers. For example, the public access modifier allows access to any code but the private access modifier restricts access to only members of a type. Other access modifiers restrict access in the range somewhere between public and private.

**Program:**

```
using System;
namespace AccessModifiers
{
    class Program
    {
        class AccessMod
        {
            public int num1;
        }
        static void Main(string[] args)
```

```
{
    AccessMod ob1 = new AccessMod();
    //Direct access to public members
    ob1.num1 = 100;
    Console.WriteLine("Number one value in main {0}",
ob1.num1);
    Console.ReadLine();
}
}
}
```

**Output:**

Number one value in main 100